



# A MetaCL Tool for Productive FPGA Programming via Automated Code Generation

PAUL SATHRE & WU-CHUN FENG

SEPT. 2, 2020



VIRGINIA TECH™

# FPGAs in HPC

- Pros:
  - High performance-per-watt
  - Flexible precision, parallelism
  - Purpose-built clusters and commodity data centers
    - Amazon EC2 FPGA Instances<sup>[1]</sup>
    - Microsoft Bing / Project Brainwave<sup>[2]</sup>
    - Project Catapult @ TACC w/ Microsoft<sup>[3]</sup> → 384 Stratix V FPGAs
    - Cygnus @ University of Tsukuba<sup>[4]</sup> → 64 Stratix 10 FPGAs
      - ARGOT: OpenCL-based GPU+FPGA cosmology/radiation code
      - Currently 401<sup>st</sup> on Top500
    - Noctua in Paderborn University, Germany<sup>[5]</sup> → 32 Stratix 10 FPGAs
- Cons:
  - Expensive, slow, and complicated development pipelines
  - Complexity of kernels is *area-constrained*, unlike traditional HPC platforms
    - synthesized hardware for kernel(s) must fit in reconfigurable area, high runtime cost to reconfigure silicon

[1] <https://aws.amazon.com/ec2/instance-types/f1/>

[2] E. Chung, et. al, "Serving DNNs in Real Time at Datacenter Scale with Project Brainwave," IEEE Micro, March 2018, Vol 38: pp. 8-20

[3] <https://www.tacc.utexas.edu/systems/catapult>

[4] <https://www.nextplatform.com/2019/04/18/supercomputer-mixes-streams-with-cpu-gpu-and-fpga/>

[5] <https://www.top500.org/news/german-university-will-deploy-fpga-powered-cray-supercomputer/>

# Why use OpenCL to program FPGAs?

Hardware Description Language (VHDL, Verilog, etc.)



Other High Level Synthesis (SystemC, Vivado HLS, etc.)



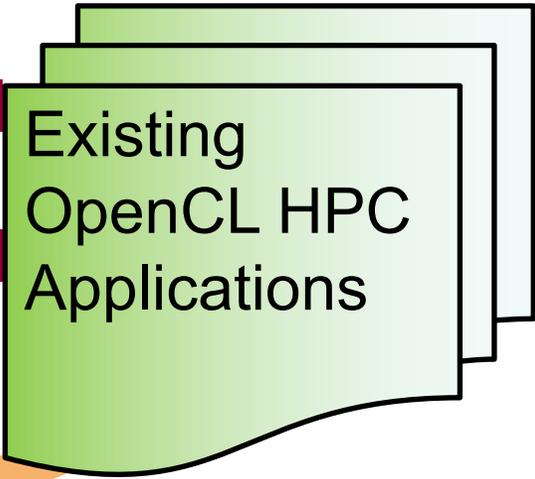
OpenCL

Host Application 

Kernels 



Existing OpenCL HPC Applications



Primary Target (dark red arrow)

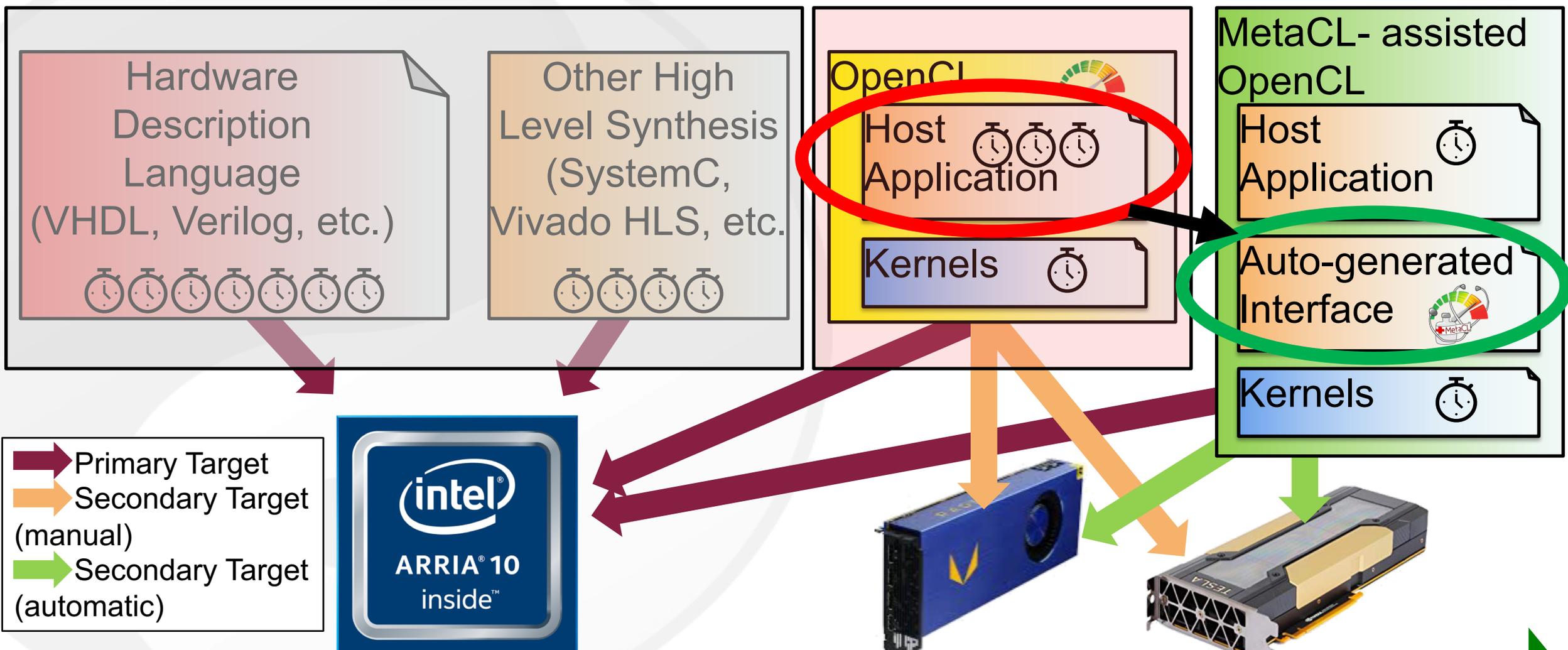
Secondary Target (manual) (orange arrow)

Secondary Target (automatic) (green arrow)

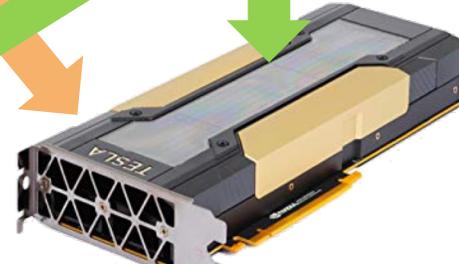


Least Programmable Most Programmable <sup>3</sup>

# How to improve on OpenCL-on-FPGA development



- Primary Target (dark red arrow)
- Secondary Target (manual) (orange arrow)
- Secondary Target (automatic) (green arrow)



Least Programmable Most Programmable 4

# Aspirations for simplified OpenCL programming

## OpenCL Kernel

```
__kernel  
void MatMul( global float* A,  
global float* B, global float* C, int  
W) {  
int tx=get_global_id(0);  
int ty=get_global_id(1);  
for(int k=0; k<W; ++k) {  
value+=A[ty*W+k]*B[k*W+tx];  
}  
C[ty*W+tx]=value;  
}
```



## OpenCL Boilerplate

```
cl_uint deviceIndex = 0;  
cl_device_id devices[MAX_DEVICES];  
unsigned numDevices = getDeviceList(devices);  
cl_device_id device = devices[deviceIndex];  
cl_context context = clCreateContext(...);  
cl_command_queue commands =  
clCreateCommandQueue(...);  
char * kernelsource = getKernelSource("matmul.cl");  
cl_program program = clCreateProgramWithSource(...);  
  
clBuildProgram(...);  
cl_kernel kernel = clCreateKernel(...);  
...  
clSetKernelArg(..., 0, ...); clSetKernelArg(..., 1, ...);  
clSetKernelArg(..., 2, ...); clSetKernelArg(..., 3, ...);  
const size_t global[2] = {W, W};  
err = clEnqueueNDRangeKernel(...);  
err = clFinish(commands);
```

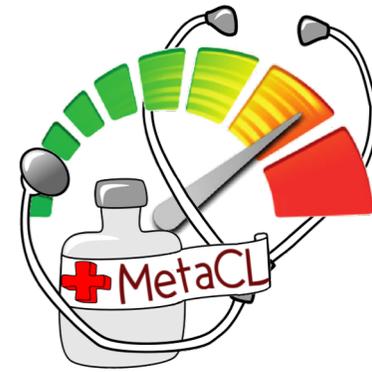


## MetaCL-assisted Boilerplate

```
meta_set_device(deviceIndex,  
metaModePreferOpenCL);  
const size_t global[3] = {W, W, 1};  
  
const size_t localsz[3] = {0, 0, 0};  
err = meta_kernel_MatMul(queue, global, localsz,  
NULL, &d_a, &d_b, &d_c, W, 1, NULL);
```



# Our Approach: Quick-n-dirty



- MetaCL: A “Meta OpenCL” host-to-kernel *interface autogenerator*
  - What does it do?
    - ✓ Parses *all* your .cl kernel implementation files
    - ✓ Generates corresponding *program* **auto-initialization** and **auto-destruction** code
    - ✓ Identifies every host-invokable kernel function
    - ✓ Generates corresponding *kernel* **auto-initialization** and **auto-destruction** code
    - ✓ Generates a launch wrapper with embedded **automatic error checking**
    - ✓ Identifies any user-defined device data types and recursively maps OpenCL API types to host equivalents to import the type to the host
    - ✓ Leverages existing MetaMorph<sup>[1]</sup> library for *platform, device, context, and command queue* **auto-management**
  - What does it not do?
    - ❑ **Parallelize or domain decompose for you**
    - ❑ **Write the kernel for you**
    - ❑ **Manage data location and coherency**

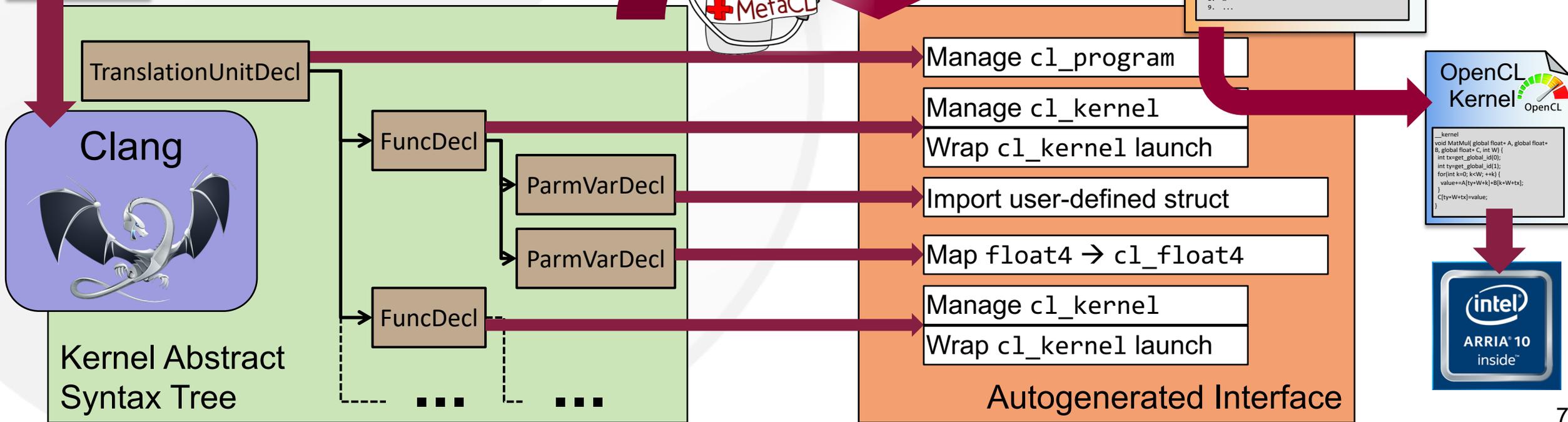
[1] Helal et al, “MetaMorph: A Library Framework for Interoperable Kernels on Multi- 6 and Many-core Clusters,” SC’16, Nov. 2016

# How it works

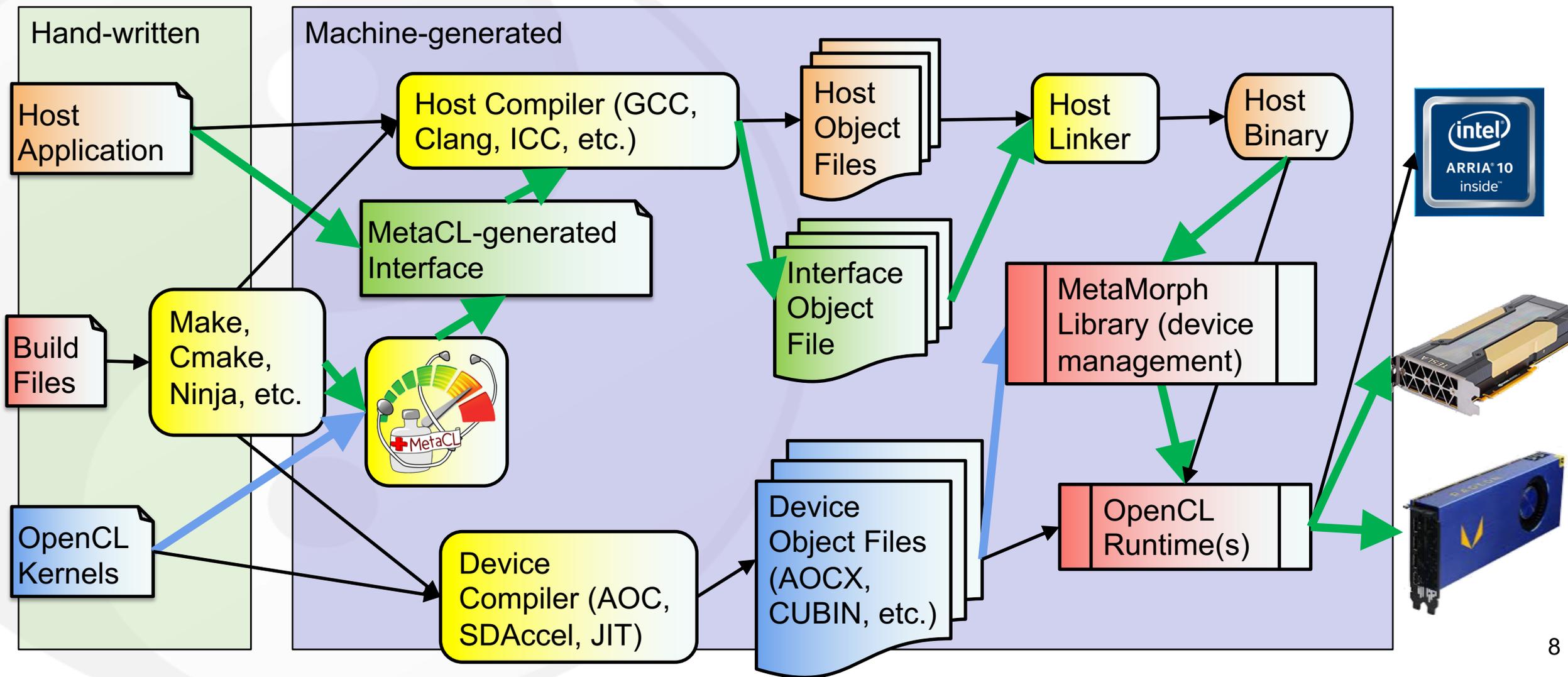
- MetaCL is a *ClangTool* → a client of the Clang compiler framework
  - Clang provides OpenCL parsing, syntax tree generation, and analysis/traversal

```
OpenCL Kernel
...
__kernel
void MatMul( global float* A, global float*
B, global float* C, int W) {
int tx=get_global_id(0);
for(int k=0; k<W; ++k) {
value+=A[ty+W*k]+B[k+W*tx];
}
C[ty+W*tx]=value;
}
```

```
Host Application
1. #include "metamorph.h"
2. #include "metacl_module.h"
3. ...
4. //Initialize MetaMorph
5. meta_set_acc(-1, metaModePreferOpenCL);
6. //Share existing OpenCL state
7. meta_set_state_OpenCL(plat, dev, ctx, queue);
8. ...
9. ...
```

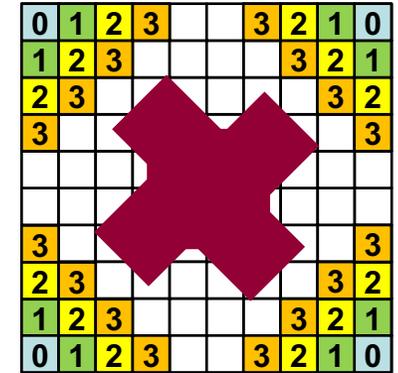


# MetaCL in a Typical Dev Toolchain



# Evaluation: “SNAP” particle transport proxy

- Proxy for LANL’s PARTISN
- Jacobi discretization that sweeps across a 3D computational domain
- All evaluation based on University of Bristol’s OpenCL+MPI implementation<sup>[2]</sup>
  - [https://github.com/UoB-HPC/SNAP\\_MPI\\_OpenCL](https://github.com/UoB-HPC/SNAP_MPI_OpenCL)
  - Utilizes a wavefront-based decomposition for parallelism across domain cells
- FPGA Complication: Arria10 area was insufficient to fit all kernels at once → divided into *inner* and *outer* c1\_programs
  - Incurs non-trivial (~1-4s) runtime cost to reconfigure the FPGA to swap
  - Stratix 10 is big enough to avoid reconfiguration → still under evaluation
  - *Generated interface is nearly identical → manual kernel fitting took minimal boilerplate refactoring*

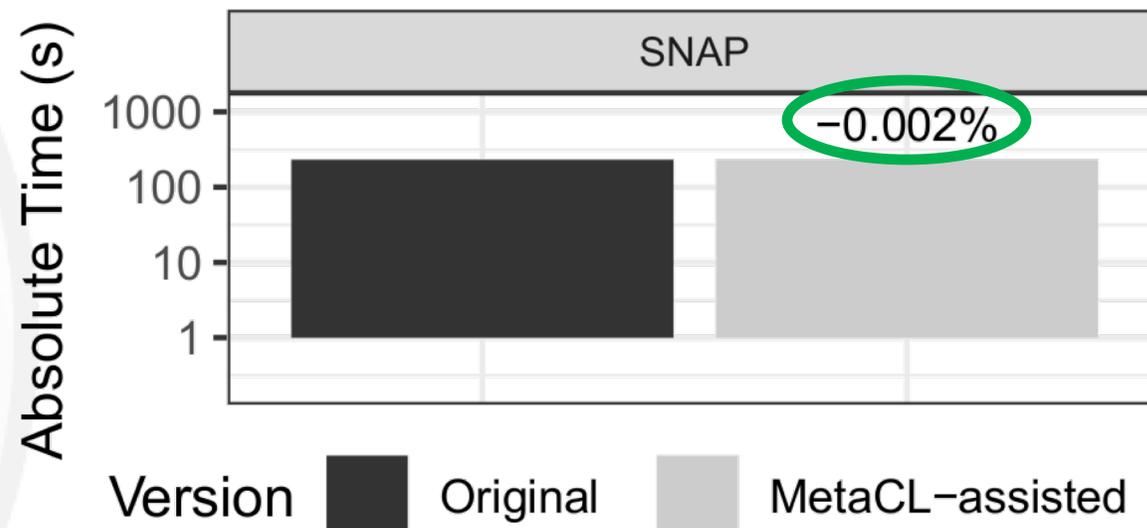


[2] T. Deakin, S. McIntosh-Smith, and W. Gaudin, “Many-Core Acceleration of a Discrete Ordinates Transport Mini-App at Extreme Scale,” in ISC’16, Jun. 2016, pp. 429–448.

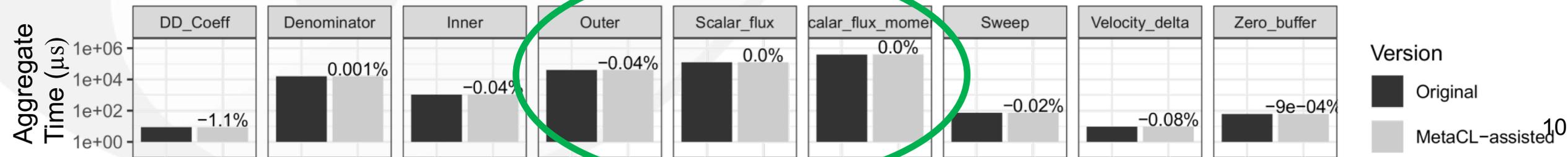
# Evaluation: Productivity with Retained Performance!

	Host Task	Original → MetaCL
OpenCL / MetaCL Boilerplate Lines	Device Management	67 → 19
	Kernel Management	184 → 69
	Data Management	59 → 59
	Total	320 → 147
	<b>Savings</b>	<b>54.1%</b>
Non-API Host Lines	Removed	43 → 0
Manually-written Host Lines	Entire Host Code	2120 → 1904
	<b>Savings</b>	<b>10.2%</b>

Whole Program Runtime (log10 scale)



SNAP Kernel Runtime (log10 scale)



# Takeaways

- MetaCL is part of open-source MetaMorph framework
  - <https://github.com/vtsynergy/MetaMorph>
  - .deb packages available, .rpms coming soon
- Eliminate over **half** of OpenCL boilerplate calls and **~10%** of manually-written host code → With **no** change in performance
- MetaCL+assisted OpenCL raises the productivity of FPGA development while providing free [functional] portability to traditional HPC platforms
- Full paper to appear at HPEC later this month

# Thanks

- Collaborators @ VT: Atharva Gondhalekar, Mohamed Hassan, Frank Wanye
- Intel DevCloud: Arria10 FPGAs and Development Tools



MetaCL and underlying components of the MetaMorph OpenCL backend have been supported in part by NSF I/UCRC CNS-1266245 via the NSF Center for High-Performance Reconfigurable Computing (CHREC) and NSF I/UCRC CNS-1822080 via the NSF Center for Space, High-performance, and Resilient Computing (SHREC).

11